18장 함수와 일급 객체

1. 일급 객체

일급 객체는 다음과 같은 조건을 만족한다.

1. 무명의 리터럴로 생성할 수 있다. 즉, 런타임에 생성이 가능하다.
2. 변수나 자료구조(객체, 배열 등)에 저장할 수 있다.
3. 함수의 매개변수에 전달할 수 있다.
4. 함수의 반환값으로 사용할 수 있다.

자바스크립트의 함수는 위의 조건을 모두 만족하므로 일급 객체다.

```
// 1. 무명의 리터럴로 생성할 수 있다.
// 2. 변수에 저장할 수 있다.
// 런타임에 함수 리터럴이 평가되어 함수 객체가 생성되고 변수에 할당된다.
const increase = function (num) {
  return ++num;
};
const decrease = function (num) {
  return --num;
};
// 2. 함수는 객체에 저장할 수 있다.
const auxs = { increase, decrease };
// 3. 함수의 매개변수에 전달할 수 있다.
// 4. 함수의 반환값으로 사용할 수 있다.
function makeCounter(aux) {
  let num = 0;

  return function () {
    num = aux(num);
    return num;
  };
}
```

```
// 3. 함수는 매개변수에게 함수를 전달할 수 있다.
const increaser = makeCounter(auxs.increase);
console.log(increaser());   // 1
console.log(increaser());   // 2
const decreaser = makeCounter(auxs.decrease);
console.log(decreaser());   // -1
console.log(decreaser());   // -2
```

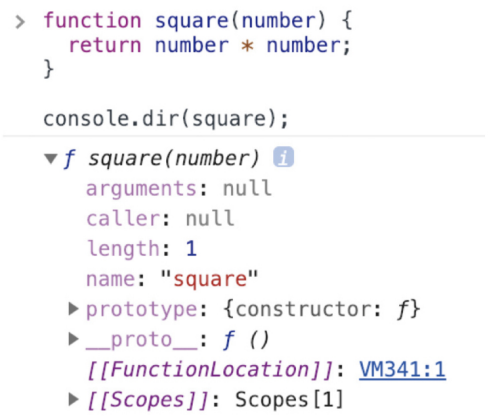
[특징]

- 함수는 객체와 동일하게 사용할 수 있으며, 객체는 값이기에 함수는 값과 동일하게 취급
  - 값을 사용할 수 있는 곳이라면 어디서든지 리터럴로 정의할 수 있으며 런타임에 함수 객체로 평가
- 일반 객체와 같이 함수의 매개변수에 전달할 수 있으며, 반환값으로 사용할 수도 있다.
- 함수는 객체이지만 일반 객체는 호출할 수 없다는 점과 달리 호출 가능하다.
- 함수 객체는 일반 객체에는 없는 **함수 고유의 프로퍼티를 소유**한다.

2. 함수 객체의 프로퍼티

```
function square(number) {
  return number * number;
}

console.dir(square);
```



https://velog.io/@kozel/모던-자바스크립트-18장-함수와-일급-객체

- 모든 프로퍼티의 프로퍼티 어트리뷰트 확인

```
function square(number) {
  return number * number;
}

console.log(Object.getOwnPropertyDescriptors(square));
/*
{
  length: {value: 1, writable: false, enumerable: false, co
nfigurable: true},
  name: {value: "square", writable: false, enumerable: fals
e, configurable: true},
  arguments: {value: null, writable: false, enumerable: fal
se, configurable: false},
  caller: {value: null, writable: false, enumerable: false,
configurable: false},
  prototype: {value: {...}, writable: true, enumerable: fal
se, configurable: false}
}
*/
/* length, arguments 등은 모두 함수 객체 고유의 데이터 프로퍼티이다.
*/

// __proto__는 square 함수의 프로퍼티가 아니다.
```

```
console.log(Object.getOwnPropertyDescriptor(square, '__prot
o__')); // undefined

// __proto__는 Object.prototype 객체의 접근자 프로퍼티다.
// square 함수는 Object.prototype 객체로부터 __proto__ 접근자 프
로퍼티를 상속받는다.
console.log(Object.getOwnPropertyDescriptor(Object.prototyp
e, '__proto__'));
// {get: f, set: f, enumerable: false, configurable: true}
```

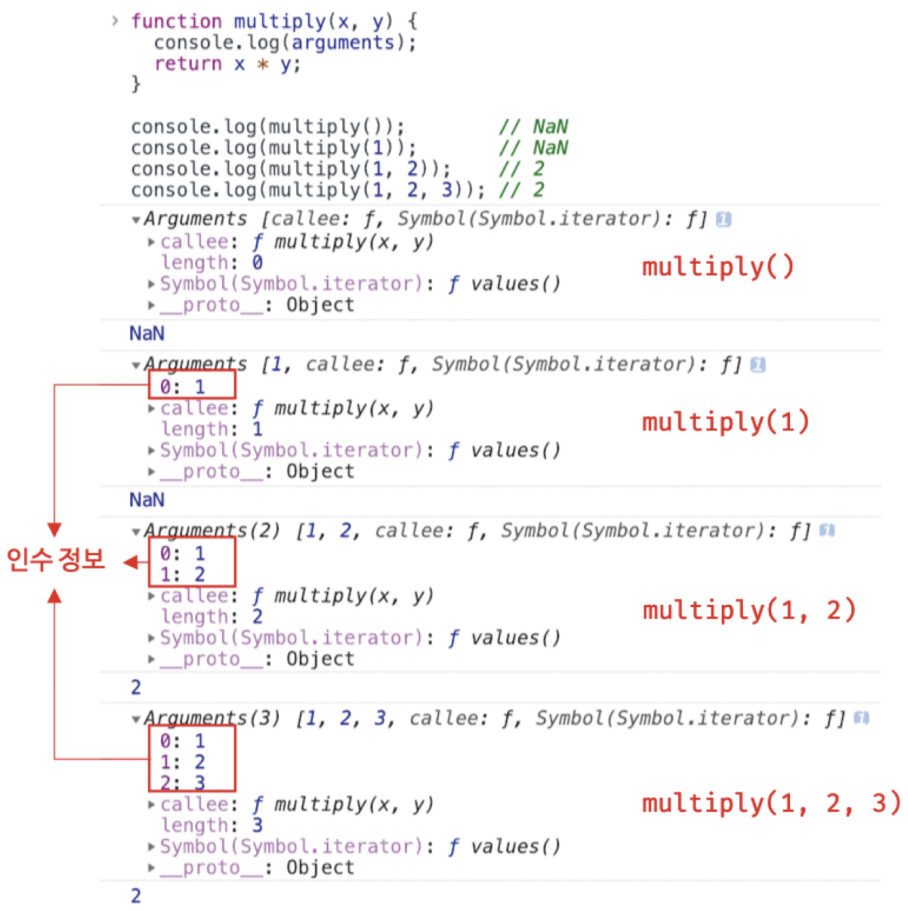
1. arguments 프로퍼티

- arguments 객체는 함수 호출 시 전달된 인수들의 정보를 담고 있는 순회 가능한 유사 배열 객체
- 함수 내부에서 지역 변수처럼 사용(외부에서는 참조 불가)
- 함수의 매개변수와 인수의 개수가 일치하는지 확인하지 않는다.
    - 호출 시 매개변수 개수만큼 인수를 전달하지 않아도 에러가 발생하지 않는다.

```
function multiply(x, y) {
  console.log(arguments);
  return x * y;
}

console.log(multiply());         // NaN
console.log(multiply(1));        // NaN
console.log(multiply(1, 2));     // 2
console.log(multiply(1, 2, 3)); // 2
```

- 함수가 호출되면 함수 몸체 내에서 암묵적으로 매개변수가 선언되고 undefined로 초기화된 이후 인수가 할당된다.
- 매개변수의 개수보다 인수를 적게 전달했을 경우 전달되지 않은 매개변수는 undefined로 초기화
- ‘’ 많이 전달했을 경우 초과된 인수는 무시
- 초과된 인수는 암묵적으로 arguments 객체의 프로퍼티로 보관



https://velog.io/@kozel/모던-자바스크립트-18장-함수와-일급-객체

💡 **arguments 객체의 Symbol (Symbol.iterator) 프로퍼티**
- arguments 객체를 순회 가능한 자료구조인 이터러블로 만들기 위한 프로퍼티

```
function multiply(x, y) {
  // 이터레이터
  const iterator = arguments[Symbol.iterator]();

  // 이터레이터의 next 메서드를 호출하여 이터러블 객체 argum
ents를 순회
  console.log(iterator.next()); // {value: 1, done:
false}
  console.log(iterator.next()); // {value: 2, done:
false}
  console.log(iterator.next()); // {value: 3, done:
false}
  console.log(iterator.next()); // {value: undefine
d, done: true}

  return x * y;
}
multiply(1, 2, 3);
```

❓ 단순히 매개변수의 value 값이 아닌 arguments 값을 확인할 때 사용해도 좋겠다

- arguments 객체는 매개변수 개수를 확정할 수 없는 가변 인자 함수를 구현할 때 유용하다.

```
function sum() {
  let res = 0;
  for (let i = 0; i < arguments.length; i++){
    res += arguments[i];
  }
  return res;
```

```
}

console.log(sum(1, 2)); // 3
```

- arguments 객체는 배열 형태로 인자 정보를 담고 있지만 실제 배열이 아닌 유사 배열 객체다.
- 유사 배열 객체 : length 프로퍼티를 가진 객체로 for 문으로 순회할 수 있는 객체
  - 배열이 아니므로 배열 메서드를 사용할 경우 에러 발생

2. caller 프로퍼티

ECMAScript 사양에 포함되지 않은 비표준 프로퍼티다.

- 함수 객체의 caller 프로퍼티는 함수 자신을 호출한 함수를 가리킨다.

```
function foo(func) {
  return func();
}
function bar() {
  return 'caller : ' + bar.caller;
}

// 브라우저에서 실행한 결과
console.log(foo(bar));  // caller : function foo(func) {...}
console.log(bar());     // caller : null
```

3. length 프로퍼티

함수 객체의 length 프로퍼티는 함수를 정의할 때 선언한 매개변수의 개수를 가리킨다.

```
function foo() {}
console.log(foo.length);    // 0

function baz(x, y){
  return x * y;
```

```
}
console.log(baz.length);    // 2
```

💡 arguments 객체의 length 프로퍼티는 **인자의 개수**를 가리키고,
함수 객체의 length 프로퍼티는
변수의 개수를 가리킨다.

4. name 프로퍼티

```
// 기명 함수 표현식
var namedFunc = function foo() {};
console.log(namedFunc.name);    // foo

// 익명 함수 표현식
var anonymousFunc = function() {};
// ES5: name 프로퍼티는 빈 문자열을 값으로 갖는다.
// ES6: name 프로퍼티는 함수 객체를 가리키는 변수 이름을 값으로 갖는다.
console.log(anonymousFunc.name);    // anonymousFunc

// 함수 선언문
function bar() {}
console.log(bar.name);  // bar

// 함수 이름과 함수 객체를 가리키는 식별자는 의미가 다르다는 것을 주의하자.
```

5. __proto__ 접근자 프로퍼티

- [[Prototype]] 내부 슬롯이 가리키는 프로토타입 객체에 접근하기 위해 사용하는 접근자 프로퍼티

```
const obj = { a: 1 };

// 객체 리터럴 방식으로 생성한 객체의 프로토타입 객체는 Object.prototyp
console.log(obj.__proto__ === Object.prototype);    // true
```

```
// 객체 리터럴 방식으로 생성한 객체는 프로토타입 객체인 Object.prototyp
// hasOwnProperty 메서드는 Object.prototype의 메서드다.
console.log(obj.hasOwnProperty('a'));           // true
console.log(obj.hasOwnProperty('__proto__'));   // false
```

6. prototype 프로퍼티

- 생성자 함수로 호출할 수 있는 함수 객체, 즉 constructor만이 소유하는 프로퍼티
- 일반 객체와 생성자 함수로 호출할 수 없는 non-constructor에는 prototype 프로퍼티가 없다.

```
// 함수 객체는 prototype 프로퍼티를 소유한다.
(function () {}).hasOwnProperty('prototype');   // true

// 일반 객체는 prototype 프로퍼티를 소유하지 않는다.
({}).hasOwnProperty('prototype');               // false
```

- prototype 프로퍼티는 함수가 객체를 생성하는 생성자 함수로 호출될 때

  생성자 함수가 생성할 인스턴스의 프로토타입 객체를 가리킨다.